1. Log in to your Sage/Cocalc account.
   
   (a) Start the Chrome browser.
   (b) Go to http://cocalc.com and sign in.
   (c) You should see an existing Project for our class. Click on that.
   (d) Click “New”, call it c30, then click “Sage Worksheet”.

2. Try the problem at http://projecteuler.net/problem=8. How will you get that number into your program??? Do not type it in by hand.

A **graph** is a mathematical object consisting of *dots* and *lines* (also called *vertices* and *edges*). Sage includes the **graphs** class which contains a number of **methods**. Some of these include constructors for making well-known graphs.

3. Try:

   ```python
   g=graphs.PetersenGraph()
   g.show()
   ```

   The *order* of a graph is the number of vertices it has. The *size* of a graph is the number of edges it has. How many vertices and edges does the Petersen graph have? Try `g.order()` and `g.size()`.

   The *order* of a graph is the number of vertices it has. The *size* of a graph is the number of edges it has.

4. Let’s get re-acquainted with paths, cycles, stars, and complete graphs. Try

   ```python
   g=graphs.PathGraph(7), g=graphs.CycleGraph(7), g=graphs.StarGraph(7), and g=graphs.CompleteGraph(7)
   ```

   To view graph `g`, use `g.show()`. The *order* of a graph is the number of vertices it has. The *size* of a graph is the number of edges it has. Find `g.order()` and `g.size()` for each graph.

5. We can create our own graph using the **Graph()** constructor. Last class we used `add_vertex()` and `add_edge()` methods. Instead of using `add_vertex()`, you can start with `Graph(5)` to get a graph with 5 vertices and no edges.

   The **type** of a graph in Sage is **Graph**. This is the **class** that the concrete graph objects like `g=graphs.PetersenGraph()` belongs to. This class contains **methods** that every concrete graph object has access to. There any numerous methods. To see the methods available to a graph `g`, just type `g.`, followed by the TAB key.
Our Own Class

The Graph class is sprawling and complicated. Now we’ll design our own class to get a feeling for the main ideas. We’ll design a general class of Dungeons and Dragons character, sample concrete character objects, methods that can be accessed by any character objects, and functions that can be used on the characters.

6. The following code defines a class called Character, together with a single method which constructs new Characters. What we have in mind is a thing (think of it as a person) that has a name.

```python
class Character():
    def __init__(self, name):
        self.name = name

Now we can create characters. Try c1=Character("John") to create a character c1 with the name “John”. We can create as many as we want. Try c2=Character("Jenn").
```

7. Our characters can’t do anything yet. So let’s add a method so they can introduce themselves.

```python
class Character():
    def __init__(self, name):
        self.name = name
    def hello(self):
        print "Hello world! I am %s." % (self.name)

Evaluate. We must create new characters in order to use the newly defined abilities. Try c3=Character("Bilbo"). Then try c3.hello()
```

8. Now let’s add attributes to our Characters: intelligence, health, strength, and fortitude. We will randomly initialize these as integers from 1 to 10. We would also like to be able to get a status report on these values. So we will add a status() method.

```python
class Character():
    def __init__(self, name):
        self.name = name
        self.intelligence=randint(1,10)
        self.health=randint(1,10)
        self.strength=randint(1,10)
        self.fortitude=randint(1,10)
    def hello(self):
        print "Hello world! I am %s." % (self.name)
    def status(self):
        print "My intelligence is %s" % (self.intelligence)
        print "My health is %s" % (self.health)
        print "My strength is %s" % (self.strength)
        print "My fortitude is %s" % (self.fortitude)

Evaluate. Let c4=Character("Frodo"). Try c4.status().
```